MATLAB is a software package for doing numerical computation. It was originally designed for solving linear algebra type problems using matrices. Its name is derived from MATrix LABoratory.

MATLAB has since been expanded and now has built-in functions for solving problems requiring data analysis, signal processing, optimization, and several other types of scientific computations. It also contains functions for 2-D and 3-D graphics and animation.
MATLAB Variable names

- Variable names are case sensitive.
- Variable names can contain up to 63 characters (as of MATLAB 6.5 and newer).
- Variable names must start with a letter and can be followed by letters, digits and underscores.

Examples:
>> x = 2;
>> abc_123 = 0.005;
>> 1ab = 2;

Error: Unexpected MATLAB expression
MATLAB Special Variables

- **pi**: Value of $\pi$
- **eps**: Smallest incremental number
- **inf**: Infinity
- **NaN**: Not a number e.g. $0/0$
- **i and j**: $i = j = \text{square root of } -1$
- **realmin**: The smallest usable positive real number
- **realmax**: The largest usable positive real number
MATLAB supports six relational operators.

- Less Than: `<`
- Less Than or Equal: `<=`
- Greater Than: `>`
- Greater Than or Equal: `>=`
- Equal To: `==`
- Not Equal To: `~=`  (NOT != like in C)
MATLAB supports three logical operators.

- **not** ~  % highest precedence
- **and** &  % equal precedence with or
- **or** |   % equal precedence with and
Matrices and MATLAB
MATLAB Matrices

- MATLAB treats all variables as matrices. For our purposes a matrix can be thought of as an array, in fact, that is how it is stored.

- Vectors are special forms of matrices and contain only one row OR one column.

- Scalars are matrices with only one row AND one column.
Generating Matrices

- A scalar can be created in MATLAB as follows:
  ```matlab
  >> x = 23;
  ```
- A matrix with only one row is called a row vector. A row vector can be created in MATLAB as follows (note the commas):
  ```matlab
  >> y = [12, 10, -3]
  y =
     12   10   -3
  ```
- A matrix with only one column is called a column vector. A column vector can be created in MATLAB as follows:
  ```matlab
  >> z = [12; 10; -3]
  z =
     12
     10
    -3
  ```
Generating Matrices

- MATLAB treats row vector and column vector very differently

- A matrix can be created in MATLAB as follows (note the commas and semicolons)

  \[
  \begin{bmatrix}
  1, 2, 3; 4, 5, 6; 7, 8, 9
  \end{bmatrix}
  \]

  \[
  \begin{array}{ccc}
  1 & 2 & 3 \\
  4 & 5 & 6 \\
  7 & 8 & 9 \\
  \end{array}
  \]

  Matrices must be rectangular!
The Matrix in MATLAB

Note: Unlike C, MATLAB's indices start from 1
A portion of a matrix can be extracted and stored in a smaller matrix by specifying the names of both matrices and the rows and columns to extract. The syntax is:

\[ \text{sub\_matrix} = \text{matrix} \left( r1 : r2, c1 : c2 \right) ; \]

where \( r1 \) and \( r2 \) specify the beginning and ending rows and \( c1 \) and \( c2 \) specify the beginning and ending columns to be extracted to make the new matrix.
Extracting a Sub-matrix

Example:

\[
X = \begin{bmatrix}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9 \\
\end{bmatrix}
\]

\[
X_{22} = X(1:2, 2:3) = \begin{bmatrix}
2 & 3 \\
5 & 6 \\
\end{bmatrix}
\]

\[
X_{13} = X(3,1:3) = \begin{bmatrix}
7 & 8 & 9 \\
\end{bmatrix}
\]

\[
X_{21} = X(1:2,1) = \begin{bmatrix}
1 \\
4 \\
\end{bmatrix}
\]
Matrix Extension

- **Matrix Extension**
  
  ```matlab
  >> a = [1,2i,0.56]
  a =
  1 0+2i 0.56
  >> a(2,4) = 0.1
  a =
  1 0+2i 0.56 0
  0 0 0 0.1
  ```

- **repmat – replicates and tiles a matrix**
  ```matlab
  >> b = [1,2;3,4]
  b =
  1 2
  3 4
  >> b_rep = repmat(b,1,2)
  b_rep =
  1 2 1 2
  3 4 3 4
  ```

- **Concatenation**
  ```matlab
  >> a = [1,2;3,4]
  a =
  1 2
  3 4
  >> a_cat = [a,2*a;3*a,2*a]
  a_cat =
  1 2 2 4
  3 4 6 8
  3 6 2 4
  9 12 6 8
  ```

  **NOTE:** The resulting matrix must be rectangular.
Matrix Addition

- Increment all the elements of a matrix by a single value

\[
\begin{bmatrix}
1 & 2 \\
3 & 4 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
6 & 7 \\
8 & 9 \\
\end{bmatrix}
\]

- Adding two matrices

\[
\begin{bmatrix}
1,2;3,4 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
7 & 9 \\
11 & 13 \\
\end{bmatrix}
\]

? Error using => plus
Matrix dimensions must agree
Matrix Multiplication

- Matrix multiplication
  
  ```
  >> a = [1,2;3,4]; (2x2)
  >> b = [1,1]; (1x2)
  >> c = b*a
  c =
  4   6
  ```

- Element wise multiplication
  
  ```
  >> a = [1,2;3,4];
  >> b = [1,½;1/3,¼];
  >> c = a.*b
  c =
  1   1
  1   1
  ```

- Error using `mtimes`
  ```
  Inner matrix dimensions must agree.
  ```
Matrix Element wise operations

- Element wise division
  ```matlab
  >> a = [1,2;1,3];
  >> b = [2,2;2,1];
  >> c = a./b
  c =
  0.5   1
  0.5   3
  ```

- Element wise multiplication
  ```matlab
  >> c = a.*b
  c =
  2    4
  2    3
  ```

- Element wise power operation
  ```matlab
  >> c = a.^2
  c =
  1    4
  1    9
  ```

  ```matlab
  >> c = a.^b
  c =
  1    4
  1    3
  ```
Matrix Manipulation functions

- **zeros**: creates an array of all zeros, Ex: \( x = \text{zeros}(3,2) \)
- **ones**: creates an array of all ones, Ex: \( x = \text{ones}(2) \)
- **eye**: creates an identity matrix, Ex: \( x = \text{eye}(3) \)
- **rand**: generates uniformly distributed random numbers in [0,1]
- **diag**: Diagonal matrices and diagonal of a matrix
- **size**: returns array dimensions
- **length**: returns length of a vector (row or column)
- **det**: Matrix determinant
- **inv**: matrix inverse
- **eig**: evaluates eigenvalues and eigenvectors
- **rank**: rank of a matrix
- **find**: searches for the given values in an array/matrix.
MATLAB inbuilt math functions
Elementary Math functions

- abs - finds absolute value of all elements in the matrix
- sign - signum function
- sin, cos, … - Trigonometric functions
- asin, acos… - Inverse trigonometric functions
- exp - Exponential
- log, log10 - natural logarithm, logarithm (base 10)
- ceil, floor - round towards +infinity, -infinity respectively
- round - round towards nearest integer
- real, imag - real and imaginary part of a complex matrix
- sort - sort elements in ascending order
Elementary Math functions

- sum, prod - summation and product of elements
- max, min - maximum and minimum of arrays
- mean, median - average and median of arrays
- std, var - Standard deviation and variance

and many more...
Graphics Fundamentals
Example 1: Plot \( \sin(x) \) and \( \cos(x) \) over \([0, 2\pi]\), on the same plot with different colours

Method 1:
```
>> x = linspace(0, 2*pi, 1000);
>> y = sin(x);
>> z = cos(x);
>> hold on;
>> plot(x, y, 'b');
>> plot(x, z, 'g');
>> xlabel 'X values';
>> ylabel 'Y values';
>> title 'Sample Plot';
>> legend ('Y data', 'Z data');
>> hold off;
```
Method 2:
```matlab
>> x = 0:0.01:2*pi;
>> y = sin(x);
>> z = cos(x);
>> figure
>> plot (x,y,x,z);
>> xlabel 'X values';
>> ylabel 'Y values';
>> title 'Sample Plot';
>> legend ('Y data','Z data');
>> grid on;
```
Example 2: Plot the following function

\[ y = \begin{cases} 
  t & 0 \leq t \leq 1 \\
  \frac{1}{t} & 1 \leq t \leq 6 
\end{cases} \]

Method 1:

```matlab
>> t1 = linspace(0,1,1000);
>> t2 = linspace(1,6,1000);
>> y1 = t1;
>> y2 = 1./t2;
>> t = [t1,t2];
>> y = [y1,y2];
>> figure
>> plot(t,y);
>> xlabel 't values', ylabel 'y values';
```
Method 2:

```plaintext
>> t = linspace(0,6,1000);
>> y = zeros(1,1000);
>> y(t()<=1) = t(t()<=1);
>> y(t()>1) = 1./ t(t()>1);
>> figure
>> plot(t,y);
>> xlabel 't values';
>> ylabel 'y values';
```
Subplots

- Syntax: subplot (rows, columns, index)

```matlab
>> subplot(4,1,1)
>> ...
>> subplot(4,1,2)
>> ...
>> subplot(4,1,3)
>> ...
>> subplot(4,1,4)
>> ...
```
Importing/Exporting Data
Load and Save

- Using load and save

```
load filename         - loads all variables from the file “filename”
load filename x      - loads only the variable x from the file
load filename a*    - loads all variables starting with ‘a’
```

For more information, type `help load` at command prompt

```
save filename        - saves all workspace variables to a binary .mat file named filename.mat
save filename x,y    - saves variables x and y in filename.mat
```

For more information, type `help save` at command prompt
Import/Export from Excel sheet

- Copy data from an excel sheet
  ```
  >> x = xlsread(filename);
  % if the file contains numeric values, text and raw data values, then
  >> [numeric, txt, raw] = xlsread(filename);
  ```

- Copy data to an excel sheet
  ```
  >> x = xlswrite('c:\matlab\work\data.xls', A, 'A2:C4')
  % will write A to the workbook file, data.xls, and attempt to fit the
  % elements of A into the rectangular worksheet region, A2:C4. On
  % success, ‘x’ will contain ‘1’, while on failure, ‘x’ will contain ‘0’.
  for more information, type help xlswrite at command prompt
  ```
Read/write from a text file

- Writing onto a text file
  ```matlab
  >> fid = fopen('filename.txt','w');
  >> count = fwrite(fid,x);
  >> fclose(fid);
  ``
  % creates a file named ‘filename.txt’ in your workspace and stores the values of variable ‘x’ in the file. ‘count’ returns the number of values successfully stored. Do not forget to close the file at the end.

- Read from a text file
  ```matlab
  >> fid = fopen('filename.txt','r');
  >> X = fscanf(fid,'%5d');
  >> fclose(fid);
  ``
  % opens the file ‘filename.txt’ which is in your workspace and loads the values in the format ‘%5d’ into the variable x.

*Other useful commands: fread, fprintf*
Flow Control in MATLAB
Flow control

- MATLAB has five flow control statements
  - if statements
  - switch statements
  - for loops
  - while loops
  - break statements
The general form of the 'if' statement is

```plaintext
>> if expression
  >> ...
  >> elseif expression
  >> ...
  >> else
  >> ...
  >> end
```

Example 1:
```plaintext
>> if i == j
  >>    a(i,j) = 2;
  >> elseif i >= j
  >>    a(i,j) = 1;
  >> else
  >>    a(i,j) = 0;
  >> end
```

Example 2:
```plaintext
>> if (attn>0.9) & (grade>60)
  >>    pass = 1;
  >> end
```
‘switch’ statement

- **switch**  Switch among several cases based on expression

- The general form of the **switch** statement is:
  ```
  >> switch switch_expr
  >>   case case_expr1
  >>      ...
  >>   case case_expr2
  >>      ...
  >>   otherwise
  >>      ...
  >> end
  ```

- Example:
  ```
  >> x = 2, y = 3;
  >> switch x
  >>   case x==y
  >>      disp('x and y are equal');
  >>   case x>y
  >>      disp('x is greater than y');
  >>   otherwise
  >>      disp('x is less than y');
  >> end
  x is less than y
  ```

  Note: Unlike C, MATLAB doesn’t need `BREAKs` in each case
'for' loop

- **for** Repeat statements a specific number of times

- The general form of a **for** statement is
  ```matlab
  >> for variable=expression
  >>   ...
  >> end
  ```

**Example 1:**
```matlab
>> for x = 0:0.05:1
   >> printf('%d
',x);
>> end
```

**Example 2:**
```matlab
>> a = zeros(n,m);
>> for i = 1:n
   >> for j = 1:m
   >>     a(i,j) = 1/(i+j);
   >> end
>> end
```
‘while’ loop

- **while** Repeat statements an indefinite number of times

- The general form of a **while** statement is

  ```matlab
  >> while expression
  >>   ...
  >> end
  ```

- **Example 1:**

  ```matlab
  >> n = 1;
  >> y = zeros(1,10);
  >> while n <= 10
  >>    y(n) = 2*n/(n+1);
  >>    n = n+1;
  >> end
  ```

- **Example 2:**

  ```matlab
  >> x = 1;
  >> while x
  >>   %execute statements
  >> end
  ```

Note: In MATLAB ‘1’ is synonymous to TRUE and ‘0’ is synonymous to ‘FALSE’
‘break’ statement

- `break` terminates the execution of `for` and `while` loops
- In nested loops, `break` terminates from the innermost loop only

Example:
```matlab
>> y = 3;
>> for x = 1:10
>>    printf('%5d',x);
>>    if (x>y)
>>        break;
>>    end
>> end
1     2     3     4
```
Efficient Programming
Efficient Programming in MATLAB

- Avoid using nested loops as far as possible
- In most cases, one can replace nested loops with efficient matrix manipulation.
- Preallocate your arrays when possible
- MATLAB comes with a huge library of in-built functions, use them when necessary
- Avoid using your own functions, MATLAB’s functions are more likely to be efficient than yours.
Example 1

Let $x[n]$ be the input to a non-causal FIR filter, with filter coefficients $h[n]$. Assume both the input values and the filter coefficients are stored in column vectors $x,h$ and are given to you. Compute the output values $y[n]$ for $n = 1,2,3$ where

$$y[n] = \sum_{k=0}^{19} h[k] x[n + k]$$
Solution

**Method 1:**

```matlab
>> y = zeros(1,3);
>> for n = 1:3
    >> for k = 0:19
      >> y(n) = y(n) + h(k) * x(n+k);
    end
  end
```

**Method 2 (avoids inner loop):**

```matlab
>> y = zeros(1,3);
>> for n = 1:3
  >> y(n) = h'*x(n:(n+19));
end
```

**Method 3 (avoids both the loops):**

```matlab
>> X = [x(1:20), x(2:21), x(3:22)];
>> y = h' * X;
```
Example 2

Compute the value of the following function

\[ y(n) = 1^3(1^3+2^3)(1^3+2^3+3^3)\cdots(1^3+2^3+\cdots+n^3) \]

for \( n = 1 \) to 20
Solution

- **Method 1:**
  ```matlab
  >> y = zeros(20,1);
  >> y(1) = 1;
  >> for n = 2:20
  >>     for m = 1:n
  >>         temp = temp + m^3;
  >>     end
  >>     y(n) = y(n-1)*temp;
  >>     temp = 0
  >> end
  >> y(n) = y(n-1)*temp;
  >> temp = 0
  >> end
  ```

- **Method 2 (avoids inner loop):**
  ```matlab
  >> y = zeros(20,1);
  >> y(1) = 1;
  >> for n = 2:20
  >>     temp = 1:n;
  >>     y(n) = y(n-1)*sum(temp.^3);
  >> end
  ```

- **Method 3 (avoids both the loops):**
  ```matlab
  >> X = tril(ones(20)*diag(1:20));
  >> x = sum(X.^3,2);
  >> Y = tril(ones(20)*diag(x)) + ...
  >>     triu(ones(20)) - eye(20);
  >> y = prod(Y,2);
  ```
Getting more help

Where to get help?

- In MATLAB’s prompt type:
  help, lookfor, helpwin, helpdesk, demos

- On the Web:
  [http://www.mathworks.com/support](http://www.mathworks.com/support)
  [http://www.math.siu.edu/MATLAB/tutorials.html](http://www.math.siu.edu/MATLAB/tutorials.html)
  [http://www.mit.edu/~pwb/cssm/](http://www.mit.edu/~pwb/cssm/)